Some notes on running ML. ML code may be written in a separate file and then loaded into ML run-time system using command use. For instance,

use ("myfile.sml");

loads the file myfile.sml in the current directory into the ML run-time system. All the function definitions from that file become available to ML system. The response to a file load is a sequence of responses to the definitions in the file. The last line must be

val it = () : unit

which indicates a successful load. You may specify a path to a file in a different directory, for instance: use("../../assign4/file1.sml"). The .sml extension is not required.

Loading files on non-Unix machines. The working directory of ML run-time system is determined when ML gets installed. The file address that you give should be relative to the working directory. To find out what the working directory is, type in

OS.FileSys.fullPath("/");

at the ML prompt. Place your file into the working directory or its subdirectory and give the relative path of the file in the command use. For instance, if your file is in the working directory, then the command use ("file1.txt") will load it.

Alternatively, you may specify absolute path names, such as

use("Desktop\file1.txt")

In this case you may position files anywhere on your computer, but the path names become longer.

Problem 1.

Write the following functions in ML using pattern-matching:

1. a recursive function append which takes two lists of the same type and appends the second list to the first. For instance,

   - append([1,2,3],[3,2,1]);
   - append([true],[false,false]);

2. a recursive function even which takes a list of any type and returns true if the list has even number of elements, false otherwise.

Problem 2. The functions for this problem and Problem 3 are defined in epoxy.mrs.umn.edu/~elenam/pub/4651/traverse.sml

The function traverse defined below works exactly as the function traverse in the assignment on Scheme.
fun traverse (combine, action, seed) = fn (nil) => seed |
(x :: xs) => combine (action x),
traverse (combine, action, seed) (xs));

Below is an example of using traverse:
val mapsquare = traverse (fn (x, y) => x :: y, fn x => x * x, nil);
mapsquare ([1, 3, ~2]);

A few things to notice in the above example:

1. the function mapsquare is defined by val definition, not by fun, because it
   is defined as the result of evaluation of an expression, whereas fun would
   require pattern-matching. Your functions should be defined the same way.

2. Unlike in Scheme, in ML * is an operator, not a function. In other words,
   * makes sense only between two numbers, not by itself, so you cannot
   pass it to a function. That's why the second parameter to traverse in
   the example above is fn x => x * x, and not just *. The same holds for
   +, -, and other operators and for ::.

3. Negative numbers in ML are written with ~, not with -, so negative 2 in
   the example above is written as ~2.

Define the following functions using traverse:

1. sumlist to add up all the elements of an integer list.

2. This problem cannot be solved using traverse, and has been
   taken out: count to count the number of elements in a list (make sure
   to test this function on a list of non-integers).

You may ignore this problem. You may instead do one or both of the
following related extra credit problems:
   - a function countInt for counting elements in a list of integers written
     using traverse
   - a recursive function count written without traverse which works
     on a list of any type (similar to append).

3. remove5 to remove all 5's from a list of integers.

4. min to find a minimum element in a list of integers Make an assumption
   about the largest number that may appear on a list. Extra credit: Is it
   possible to write a function betterMin (as defined in the problem set 2)
   in ML? Please explain your answer.

5. reverseInt to reverse a list of integers. Note that ML type system doesn't
   allow a general reverse function which will work on a list of any type. You
   need to specify the type of at least one parameter to traverse in order
   to make reverse work. For instance, fn x : int => x is an identity
   function on integers.

You may use append from problem 1 for this question.
You may define auxiliary functions if you find them helpful. Note that “do” is a reserved word in ML which can’t be used as a parameter name. That’s why the second parameter of traverse is renamed to action.

Problem 3. You are given the following datatype ‘a tree (defined in the file epoxy.mrs.mcs.unl.edu/~elena/pub/4651/tree.sml):
  datatype ‘a tree = LEAF of ‘a | NODE of ‘a tree * ‘a tree;
The file also contains the following instances of this datatype:
  val intTree1 = LEAF (5);

  val intTree2 = NODE (NODE (NODE (LEAF (3), LEAF (4)), LEAF (6)),
  NODE (NODE (LEAF (5), NODE (LEAF (3), LEAF (5))), LEAF (0)));

  val strTree1 = LEAF ("apples");

  val strTree2 = NODE (NODE (NODE (LEAF ("apples"), LEAF ("bananas")),
  LEAF ("oranges")),
  NODE (NODE (LEAF ("grapes"), NODE (LEAF ("pears"), LEAF
  ("apples")))), LEAF ("watermelons"));

  As an example of working with trees in ML, consider the following function:
  fun addtree (LEAF n) = n
  addtree (NODE (t1, t2)) = addtree (t1) + addtree (t2);

  Question 1. Draw a picture of intTree2 as a binary tree.

  Question 2. Write a function traversetree which takes two parameters combine and action so that action is a function applied to the leaf and combine is used to combine the results for the two subnodes of a node. For instance, the addtree function above can be written, using traversetree, as
  val addtree = traversetree (fn (x, y) => x + y, fn x => x);

  Question 3. Using the traversetree, define the following functions:

  1. concat which concatenates all elements of a string tree separated by spaces. For instance, the results for the trees strTree1 and strTree2 would be as follows:

     val it = "apples" : string
     val it = "apples bananas oranges grapes pears apples watermelons" : string

  2. flattenint and flattenstr which make a list of all elements of a tree. For instance, flattenint (intTree2) results in

     val it = [3,4,6,5,3,5,0] : int list

  Note that, just like in reverse in Problem 2, you need separate functions for a tree of integers and for a tree of strings. You may use append for this problem.